```lua
--         ___                ___
--        /\_ \              /\_ \
--  _ __  \//\ \             \//\ \    __  __     __
-- /\`'__\  \ \ \              \ \ \  /\ \/\ \  /'__`\
-- \ \ \/    \_\ \_    __      \_\ \_\ \ \_\ \/\ \L\.\_
--  \ \_\    /\____\  /\_\     /\____\\ \____/\ \__/.\_\
--   \/_/    \/____/  \/_/     \/____/ \/___/  \/__/\/_/
--
local l = require"lib"
local the = l.settings[[

RL.LUA : stings
(c)2022 Tim Menzies <timm@ieee.org> BSD(2clause).

USAGE:
  lua rlgo.lua -[bghk] [ARG]

OPTIONS:
 -b  --bins   discretization control = 8
 -F  --Far    in "far", how far to seek = .95
 -g  --go     start-up action        = pass
 -h  --help   show help              = false
 -k  --keep   keep only these nums   = 256
 -s  --seed   random number see      = 10019
 -S  --Some   in "far", how many to search = 512]]

local About= {} -- factor for making columns
local Col  = {} -- summarize one column
local Data = {} -- store rows, and their column summaries
local Row  = {} -- store one row

-- CODE CONVENTIONS
-- Leading__upper_case : class
-- i.                  :  instance va
-- l. s                : reference to a library function
-- prefix _            : some internal function,variable.
--
-- type hints: where practical, on function arguments,
--    - t = table
--    - prefix s=string
--    - prefix n=num
--    - prefix is=boolean
--    - class names in lower case denote vars of that class
--    - suffix s denotes table of things

--------------------------------------------------------------------------------
-- .__..          ,
-- [__]|_  _ . .-+-
-- |  |[_)(_)(_| |
--
function About.new(sNames)
  return About._cols({names=sNames, all={}, x={}, y={}, klass=nil},sNames) end

-- How to recognize different column types
local _is={
  nom   = "^[a-z]",  -- ratio cols start with uppercase
  goal  = "[!+-]$",  -- !=klass, [+,-]=maximize,minimize
  klass = "!$",       -- klass if "!"
  skip  = ":$",       -- skip if ":"
  less  = "-$"}       -- minimize if "-"

-- Turn a list of column names into Col objects. If the new col is independent
-- or dependent or a goal attribute then remember that in i.x or i.y or i.klass.
function About._cols(i,sNames)
  for at,name in pairs(sNames) do
    local col = l.push(i.all, Col.new(name,at))
    if not name:find(_is.skip) then
      l.push(name:find(_is.goal) and i.y or i.x, col)
      if name:find(_is.klass) then i.klass=col end end end
  return i end

-- Update, only the non-skipped cols (i.e. those found in i.x and j.x.
function About.add(i,t)
  local row = t.cells and t or Row.new(i.about, t)
  for _,cols in pairs{i.x,i.y} do
    for _,col1 in pairs(cols) do
      Col.add(col1, row.cells[col1.at]) end end
  return row  end

--------------------------------------------------------------------------------
-- .__
-- [__) _ .    ,
-- |  \(_) \/\/
--
-- Hold one record
function Row.new(about,t)
  return {_about=about, cells=t, cooked=l.map(t,l.same)} end

-- Everything in rows, sorted by distance to i.
function Row.around(i,rows)
  local fun = function(j) return {row=j, d=Row.dist(i,j)} end
  return l.sort(l.map(rows, fun), lt"d") end

-- Recommend sorting i before j (since i is better).
function Row.better(i,j)
  i.evaled,j.evaled= true,true
  local s1,s2,d,n,x,y=0,0,0,0
  local ys,e = i._about.y,math.exp(1)
  for _,col in pairs(ys) do
    x,y= i.cells[col.at], j.cells[col.at]
    x,y= Col.norm(col,x), Col.norm(col,y)
    s1 = s1 - e^(Col.w * (x-y)/#ys)
    s2 = s2 - e^(Col.w * (y-x)/#ys) end
  return s1/#ys < s2/#ys end

-- Distance
function Row.dist(i,j)
  local d,n,x,y,dist1=0,0
  local cols = cols or i._about.x
  for _,col in pairs(cols) do
    x,y = i.cells[col.at], j.cells[col.at]
    d   = d + Col.dist(col,x,y)^the.p
    n   = n + 1 end
  return (d/n)^(1/the.p) end


--------------------------------------------------------------------------------
--  ___    .
-- /   ` _ |
-- \__.(_)|
--
-- Summarize one column.
function Col.new(txt,at)
  txt = txt or ""
  return {n    = 0,             -- how many items seen?
          at   = at or 0,         -- position ot column
          txt  = txt,            -- column header
          isNom= txt:find(_is.nom),
          w    = txt:find(_is.less) and -1 or 1,
          ok   = true,             -- false if some update needed
          _has  = {}} end           -- place to keep (some) column values.

-- Update
function Col.add(i,x)
  if x ~= "?" then
    i.n = i.n + 1
    if   i.isNom
    then i._has[x] = 1 + (i._has[x] or 0)
    else local pos
         if     #i._has  < the.keep     then pos=  1 + (#i._has)
         elseif l.rand() < the.keep/i.n then pos=l.rand(#i._has) end
         if pos then
           i.ok=false -- kept items are no longer sorted
           i._has[pos]=x end end end end

-- Distance
function Col.dist(i,x,y)
  if x=="?" and y=="?" then return 1 end
  if   i.isNom
  then return  x==y and 0 or 1
  else if x=="?" and y=="?" then return 1 end
       if     x=="?" then y = Col.norm(i,y); x=y<.5 and 1 or 0
       elseif y=="?" then x = Col.norm(i,x); y=x<.5 and 1 or 0
       else   x,y = Col.norm(i,x), Col.norm(i,y) end
       return math.abs(x-y) end end

-- Diversity
function Col.div(i)
  if   i.isNom
  then local e=0
       for _,v in pairs(i._has) do
         if v>0 then e=e-v/i.n*math.log(v/i.n,2) end end
       return e
  else local t=Col.has(i)
       return (l.per(t,.9) - l.per(t,.1))/2.56 end end

-- Sorted contents
function Col.has(i)
  if i.isNom then return i._has end
  if not i.ok then table.sort(i._has) end
  i.ok=true
  return i._has end

-- Central tendency
function Col.mid(i)
  if   i.isNom
  then local mode,most=nil,-1
       for k,v in pairs(i._has) do if v>most then mode,most=k,v end end
       return mode
  else return l.per(Col.has(i),.5) end end

-- Return num, scaled to 0..1 for lo..hi
function Col.norm(i,num)
  local a= Col.has(i) -- "a" contains all our numbers,  sorted.
  return a[#a] - a[1] < 1E-9 and 0 or (num-a[1])/(a[#a]-a[1]) end

-- Map x to a small range of values.
function Col.discretize(i,x,     a,b,lo,hi)
  if i.isNom then return x else
    a = has(i)
    lo,hi = a[1], a[#a]
    b = (hi - lo)/the.bins
    return hi==lo and 1 or math.floor(x/b+.5)*b  end end


--------------------------------------------------------------------------------
-- .__     ,
-- |  \ _.-+- _.
-- |__/(_] | (_]
--
-- Holds n records
function Data.new(t) return {rows={}, about=About.new(t) } end

-- Update
function Data.add(i,t) l.push(i.rows, About.add(i.about,t)) end

-- Replicate structure
function Data.clone(i,  t)
  local out = Data.new(i.about.names)
  for _,row in pairs(t or {}) do Data.add(data,row) end
  return data end

-- Discretize all row values (writing those vals to "cooked").
function Data.discretize(i)
  for _,col in pairs(i.about.x) do
    for _,row in pairs(i.rows) do
      local x = row.cells[col.at]
      if x~= "?" then
        row.cooked[col.at] = discretize(col,x) end end end end

-- Recursively bi-cluster one Data into sub-Datas.
function Data.cluster(i,  rowAbove,stop)
  stop = stop or (#i.rows)^the.Min
  if #i.rows >= 2*stop then
    local A,B,As,Bs,c = Data.half(i.rows,rowAbove)
    i.halves = {c=c, A=A, B=B,
                kids = { Data.cluster(Data.clone(i,As), A, stop),
                         Data.cluster(Data.clone(i,Bs), B, stop) }}end
  return i end

-- Split data according to distance to two  distant points A,B
-- To speed things up, find distant points via A=far(any()) and B=far(A).
-- To speed things up, try to reuse a distant point from above (see rowAbove).
-- To speed things up, only look at some of the rows (see the.Some).
-- To dodge outliers, don't search all the way to edge (see the.Far).
function Data.half(i, rows,  rowAbove)
  local some= l.many(rows, the.Some)
  local function far(row)
    return l.per(Row.around(row,some), the.Far).row end
  local function project(row)
    local a,b = Row.dist(row,A), Row.dist(row,B)
    return {row=row, x=(a^2 + c^2 - b^2)/(2*c)} end
  local A= rowAbove or far(l.any(some))
  local B= far(A)
  local c= Row.dist(A,B)
  local As,Bs = {},{}
  for n,rowx in pairs(l.sort(l.map(rows, project),l.lt"x")) do
    push(n < #rows/2 and As or Bs, rowx.row) end
  return A,B,As,Bs,c end

-- Load from file
function Data.load(sFilename,        data)
  l.csv(sFilename, function(row)
    if data then Data.add(data,row) else data=Data.new(row) end end)
  return data end

-- Central tendancy
function Data.mid(i) return l.map(i.about.y, Col.mid) end

-- Guess the sort order of the rows by peeking at a few distant points.
function Data.optimize(i,  rowAbove,stop,out)
  stop = stop or (#i.rows)^the.Min
  out = out or {}
  if   #i.rows < 2*stop
  then for _,row in pairs(i.rows) do push(out,row) end
  else local A,B,As,Bs,c = Data.half(i.rows, rowAbove)
       if Row.better(A,B)
       then for j=#Bs,1 do push(out,Bs[j]) end
            Data.optimize(Data.clone(i,rev(As)), A, stop, out)
       else for _,row in pairs(As) do push(out,row) end
            Data.optimize(Data.clone(i,Bs), B, stop, out)
       end end
  return out end

--------------------------------------------------------------------------------
return {Data=Data,Row=Row,Col=Col,About=About,the=the}
```